On the parallelization of slice-based Keccak implementations on Xilinx FPGAs

Jori Winderickx*, Joan Daemen† and Nele Mentens*
*KU Leuven, ESAT/COSIC & iMinds, Leuven, Belgium
†STMicroelectronics Belgium & Radboud University, the Netherlands

***Abstract*—This work explores the parallelization of slice-based lightweight Keccak implementations. The functionality of Xilinx FPGAs to use a single slice as a 32-bit Shift Register Lookup table (SRL) was recently used by Winderickx et al. [1] to implement Keccak. This implementation resulted in the smallest Keccak implementation, given that a custom interface was used. To enhance the implementation, we explore the parallelization of the datapath. Four datapath widths are used: 25, 50, 100 and 200 bits.**

The implementations with a datapath width of 25 and 50 bits give better area and throughput results than other slice-based lightweight Keccak implementations; larger datapath widths result in an inefficient usage of the SRL functionality. Furthermore, the custom interface with datapath widths larger than 50 bits is not compatible with 64-bit microprocessors and is therefore unusable in practical scenarios. The standard-compliant implementations perform worse than other slice-based lightweight Keccak implementations for all datapath widths.

I. Introduction

The Keccak algorithm [2] was the winner of the SHA-3 competition, organized by the National Institute of Standards and Technology (NIST). To use this algorithm in embedded devices, many lightweight implementations were explored [1], [3]–[6]. In this paper we examine the parallelization of the lightweight implementation proposed by Winderickx et al. in [1].

The implementation of [1] uses a slice-based method to implement Keccak. Furthermore, it utilizes the 'Shift Register Lookup Table (SRL)' function of the SLICEM slices in Xilinx FPGAs. This results in the smallest implementation of Keccak, given that a custom interface is used. The paper provided two solutions: a custom interface and a standard-compliant interface. The custom interface utilizes one slice of the Keccak state as input and output, while the standard-compliant interface utilizes a 64-bit data input/output, corresponding to a lane in the Keccak state.

This paper explores the parallelization of the two solutions provided by Winderickx et al. [1]. Via parallelization of the datapath, we can enhance the throughput and examine the impact on the resource utilization in Xilinx FPGAs. All implementations are compared with the results of Jungk et al. in [4] and Jungk in [6].

To make a fair comparison with the other implementations, a Virtex-5 FPGA is used. The target board is the ML505 board. Xilinx ISE 14.7 is used for the evaluation, mapping, routing, etc. of the implementations on the FPGA.

This paper will start with providing information on Keccak and the utilized SRL technique in Section II and the discussion on the the related work in Section III. Then the architecture and the parallelization is explained in Section IV. Lastly, the results are discussed in Section V and a conclusion is given in Section VI.

II. Preliminaries

A. Keccak

The implementation evaluated is based upon the Keccak-f[1600] permutation [7]. The Keccak-f[1600] permutation is one of the 7 permutations defined by Keccak, indicated by Keccak-f[b]. Its round function R is composed of 5 step mappings: θ, ρ, χ, π and ι (respectively Theta, Rho, Chi, Pi and Iota). Furthermore these operate on a 3-dimensional state, denoted by $a[x][y][z]$. The Keccak-f[1600] contains 24 iterated rounds. The architecture that is analysed in this paper uses an optimized implementation of Rho and Theta to achieve an area-efficient implementation. The five step mappings are here defined:

- $\theta : a[x][y][z] \leftarrow a[x][y][z] + \sum_{y'=0}^{4} a[x-1][y'][z] + \sum_{y'=0}^{4} a[x+1][y'][z-1]$,
- $\rho : a[x][y][z] \leftarrow a[x][y][z-(t+1)(t+2)/2]$,

  with t satisfying $0 \leq t < 24$

  and $\begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}^t \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix}$ in GF$(5)^{2\times 2}$,

  or $t = -1$ if $x = y = 0$.
- $\pi : a[x][y] \leftarrow a[x'][y']$, with $\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} x' \\ y' \end{pmatrix}$
- $\chi : a[x] \leftarrow a[x] + (a[x+1]+1)a[x+2]$
- $\iota : a \leftarrow a + RC[i_r]$

The analysed implementation follows a slice-based approach of Keccak. This means that the combinatorial logic of the implementations operates on slices. A representation of the state, slice and lane is presented in Fig. 1. The Rho implementation combines the Rho step, essentially rotations

within lanes, and the storage of the state. The rotation is depended on the offset of the specific lane in the state, presented in Table I.

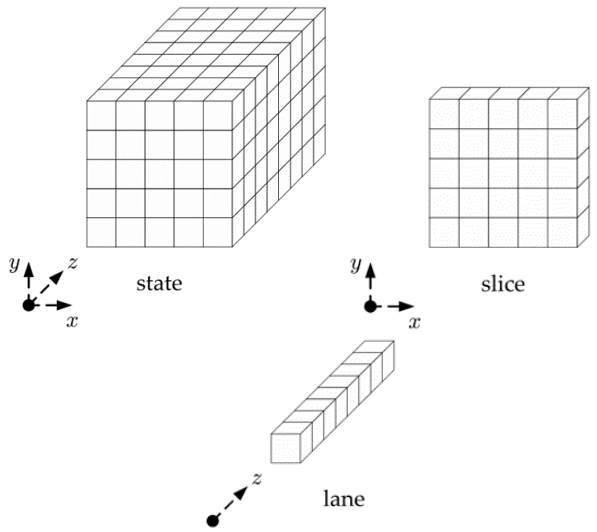

Fig. 1. Representation of the state, slice and lane in the Keccak-f[b] permutation

The length of the shift register needs to be 64 bits to store the state and the additional offset to incorporate the Rho step. The length l of the lane with coordinates x and y is defined by: $l(x, y) = 64 + (\text{offset}(x, y) \mod 64)$. The modulo calculation is used to map the offsets onto the size of the state.

TABLE I
THE OFFSETS OF ρ

	x=3	x=4	x=0	x=1	x=2
y = 2	153	231	3	10	171
y = 1	55	276	36	300	6
y = 0	28	91	0	1	190
y = 4	120	78	210	66	253
y = 3	21	136	105	45	15

B. Shift Register Lookup table (SRL)

The configurable logic of the Xilinx Virtex-5 FPGA is made up of configurable logic blocks (CLBs). The CLBs consist of slices. These slices are the reconfigurable building blocks of the FPGA and should not be confused with the slices in the Keccak state. Therefore, whenever we refer to a slice in the FPGA, we use the term reconfigurable slice in the remainder of the paper. In the Virtex-5 FPGA, the reconfigurable slices of type SLICEM contain 4 Lookup Tables (LUTs) and 4 flip-flops. In this type of reconfigurable slice, the LUTs can be configured into a 32-bit shift register, called a Shift Register Lookup table (SRL), without using the flip-flops. The schematic representation is displayed in Fig. 2. The SRL has at maximum two outputs: a selectable output Q and the last bit or shift-out output (Q31). The selectable output can be configured to output one of the 32 bits in the shift register by using the corresponding address. The inputs of the SRL consist of the clock (CLK) and write-enable (WE) input to control the flow and the shift-in input (D).

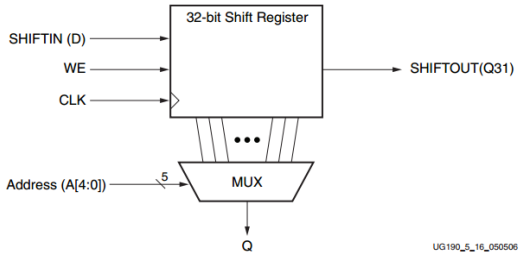

Fig. 2. Schematic representation of a Shift Register Lookup table on Virtex-5 FPGA

III. Related work

The work done by Jungk et al. [4] analyses possible design strategies for Keccak. To evaluate the design strategies, a generic slice-based architecture of Keccak was created. This architecture is generic in the sense that it is parameterized with relation to the data path width, the state size and the size of the message digest. The state is implemented in distributed RAM on the FPGA. Furthermore the architecture uses two phases, the absorption phase and the processing phase. The absorption phase is separated because input needs to be loaded into the state before the actual Keccak processing can commence. The results of their slice-based Keccak implementations are among the smallest published and are therefore used in this paper as comparison. In the implementations of Winderickx et al. [1], the SRL function of the SLICEM reconfigurable slices is used to efficiently store the state. Furthermore it introduces the use of a custom interface that does not require an absorption phase. The absorption phase is however similar to the reordering of input from lanes to slices of the standard compliant interface implementation of Winderickx et al.

The PhD dissertation of Bernhard Jungk [6] proposes further optimizations on his earlier work [3]. However, in his dissertation, he did not report the results of the throughput without absorption. The results of his dissertation are therefore added as a different architecture in the comparison.

IV. SRL-oriented architecture

The non-parallelized architecture of the SRL-based implementation [1] is shown in Fig. 3. It consists of a loop-structured datapath and a FSM that controls the flow. The datapath contains the five steps of the Keccak-f round function. It uses one slice per cycle to calculate the next state of the Keccak-f permutation. Currently 64 cycles are needed to calculate one round. The 64 cycles correlate to the 64 slices that the state is comprised of, the state of 1600 bits contains 64 slices of 25 bits.

A. Parallelization

To evaluate the parallelization of the SRL slice-based implementation, we created a generic VHDL implementation of the non-parallelized architecture, shown in Fig. 4. This

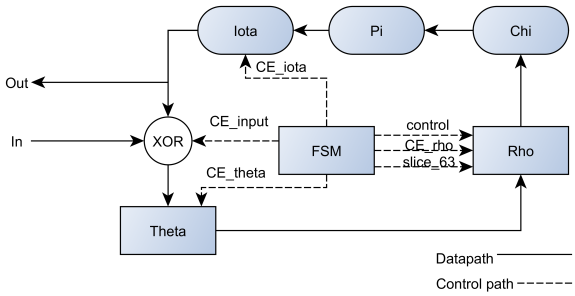

Fig. 3. Schematic of the non-parallelized SRL-oriented Architecture

generic VHDL implementation is parameterized to facilitate the parallelization. In this paper, we analyze four different levels of parallelization, namely $n = 1, 2, 4, 8$, corresponding to datapath widths of 25, 50, 100 and 200. This has a large impact on the cycles needed for a Keccak-f permutation. During each cycle, n slices are processed, where n corresponds to the level of parallelization. This results in a cycle count of $64/n$ cycles for one round of the Keccak-f permutation.

For each slice in the datapath, a Iota, Pi and Chi implementation is added because they are in-slice calculations. The size of the Iota, Pi and Chi implementations should scale linear with the parallelization of the datapath. The Theta and Rho implementations are not in-slice calculations and do not scale as in the Iota, Pi and Chi case. The implementations of Rho and Theta are explained in the following sections.

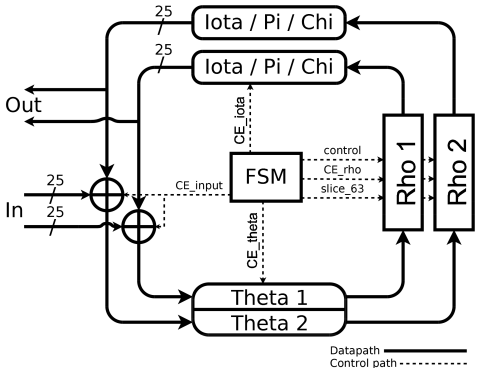

Fig. 4. Schematic of the parallelized SRL-oriented Architecture, datapath is doubled

B. Implementation of Rho

The SRLs are used to implement the shift registers for the implementation of Rho. In the non-parallelized implementation, one shift register is used for each lane and one slice per cycle is shifted into these lane registers. Each lane register has a different length so that the permutation of Rho can be emulated. In the case of parallelization, the lane registers need to be divided into multiple shift registers for each lane, shown in Fig. 5. The amount of SRLs per lane corresponds to the size of the datapath, i.e. the amount of slices calculated at once (n). Each shift register has its own input, outputs and control lines to select the right output bit. In case of $n = 2$, each lane contains two shift registers. If the length of the lane register is divisible by n, the divided shift registers are of equal length, an example is the lane 0,0 in Fig. 5. If the length is not divisible by n, the remainder of the length, $r = shift\_length \mod n$, is spread out over the first few shift registers. This results also in a remap of the input onto the shift registers, e.g. lane 0,1 in Fig. 5.

With each level of parallelization, the implementation will require more and smaller shift registers. This results in more SRLs and therefore also in a larger implementation.

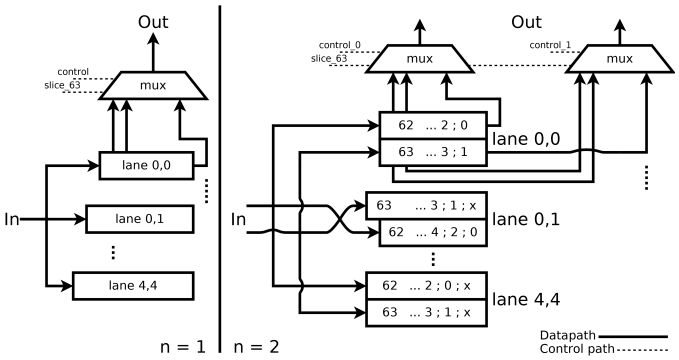

Fig. 5. Schematic of the Rho implementation with no parallelization (n = 1) and based on a double-width datapath (n = 2)

C. Implementation of Theta

In the non-parallelized implementation, a 5-bit buffer register was used to buffer the column parity of the slice that is treated in the last cycle of the series of cycles that compute one round. The parallelized implementation also needs only one 5-bit buffer. The column parity of the last slice in the datapath is calculated and stored in the 5-bit register. It can then be used in the next cycle to calculate the Theta output of the first slice in the datapath, depicted in Fig. 6.

For each slice in the datapath a column parity and Theta block is added. The Theta implementation should therefore scale proportionally to the level of parallelization.

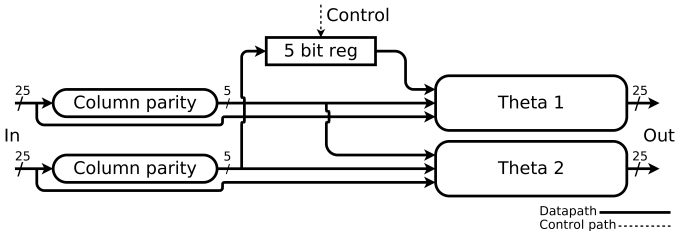

Fig. 6. Schematic of the Theta mapping in the parallel implementation, datapath is doubled

D. Interface buffers

The interface buffers are used to reorder the input from lanes to slices. In the non-parallelized implementation, large multiplexers are needed to select only one slice. By parallelizing the implementation, the multiplexers will shrink because

a less precise selection needs to be made. For parallelization level $n = 1$, each slice needs to be addressable and the multiplexer therefore requires 64 selections. For parallelization level $n = 8$, 8 slices are required each cycle and the multiplexer therefore has 8 selections. The rest of the buffer implementation stays the same. 8 dual-port BRAMs are used to store the input and 4 dual port BRAMs are used as shift registers to cache the slice output.

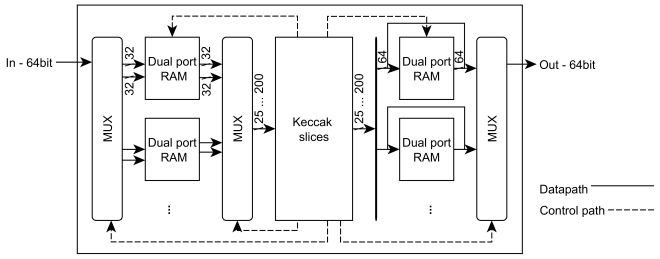

Fig. 7. Parallelization of interface buffers

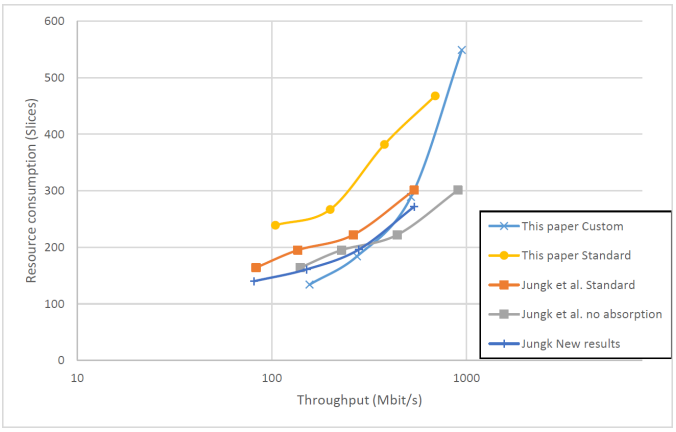

Fig. 8. Size en throughput comparison of the parallelization of lightweight Keccak-f implementations

V. Results

The results of our parallelized implementations are compared in Table. II. The 'Jungk New results' architecture represents the results of Jungk. [6]. The 'Jungk et al.' architecture contains the results of Jungk et al. [4] with and without the absorption phase (no absorption (n.a.)). The graph in Fig. 8 visualizes the results. We compare, in the graph, the size and the throughput by comparing the total amount of reconfigurable slices and the data throughput of one complete Keccak-f permutation, respectively.

Our parallelized lightweight Keccak implementations do not scale as well in size as the implementation of [4] and [6]. For the first two levels of parallelization, with datapath widths of respectively 25 and 50 bits, our SRL-based implementations with the custom interface occupy the least resources. However, with each parallelization level, the amount of shift registers grows in order to provide multiple shift registers for each lane. There are now more but shorter shift registers for the storage of the Keccak state. More shift registers result in a much less efficient implementation because the amount of SRLs will rise when the width of the datapath grows. As expected, the implementation with a standard-compliant interface consumes more resources than the implementations of [4] and [6], but we notice that a datapath width of 200 bits results in a standard-compliant implementation that is smaller than the implementation with the custom interface. The first reason is that less multiplexers are required for each larger step of the datapath causing the buffer implementation to shrink in size and secondly the software needs to handle more interface's inputs and outputs with the custom interface implementation than with the standard compliant implementation.

The throughput of our SRL-based implementations scale better than the other implementations, even if 'no absorption' is taken into account. The SRL-based implementation suffers less from the buffering of the input than the absorption phase of the implementation in [4]. The SRL-based implementations with the standard-compliant interface do not scale as well as the custom interface approach because they require additional buffering.

TABLE II
Implementation results on Virtex-5 FPGA of parallelized Keccak architectures

Datapath	Architecture	Slices	Freq (MHz)	T.put (Mbps)	T.put n.a. (Mbps)	Efficiency (Mbit/s/slice)
25	This paper Custom	134	248	156		1.17
	This paper Standard	239	168	104		1.49
	Jungk et al.	164	206	83	140	0.51
	Jungk New results	140	200	81		0.58
50	This paper Custom	184	221	274		1.49
	This paper Standard	267	165	200		0.75
	Jungk et al.	195	168	136	228	0.69
	Jungk New results	161	186	151		0.93
100	This paper Custom	289	215	518		1.79
	This paper Standard	382	165	379		0.99
	Jungk et al.	222	162	262	441	1.18
	Jungk New results	196	173	280		1.43
200	This paper Custom	549	208	947		1.72
	This paper Standard	468	165	691		1.48
	Jungk et al.	301	166	538	903	1.79
	Jungk New results	272	166	539		1.98

VI. Conclusion

We explore different versions of slice-based Keccak implementations on a Xilinx Virtex-5 FPGA. Both a custom interface and a standard-compliant interface are implemented for four different datapath widths. For datapath widths of 25 and 50 bits, the custom interface implementation is more efficient than the implementation of Jungk et al. [4] and Jungk [6]. For larger datapath widths, our implementations are less beneficial in terms of resource occupation than the implementations in [4] and [6]. Moreover, the utilization of a custom interface for datapath widths of 100 and 200 bits is not compatible with 64-bit microprocessors, requiring a work-around to enable communication with the FPGA coprocessor.

ACKNOWLEDGMENT

This work is a result of the CORNET project "DynamIA: Dynamic Hardware Reconfiguration in Industrial Applications"; it is funded by IWT Flanders with reference number 140389. In addition, this work was supported by the Flemish Government, FWO G.0550.12N, G.00130.13N, FWO G.0876.14N and Thresholds G0842.13 and by the Hercules Foundation AKUL/11/19, and through the Horizon 2020 research and innovation programme under grant agreement 644052 HECTOR.

References

[1] J. Winderickx, N. Mentens, and J. Daemen, "Exploring the use of shift register lookup tables for keccak implementations on xilinx fpgas," *Accepted at 26th International Conference on Field-Programmable Logic and Applications, FPL 2016*, p. 4, 2016.

[2] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche, "The Keccak SHA-3 submission," January 2011, http://keccak.noekeon.org/.

[3] B. Jungk and J. Apfelbeck, "Area-efficient FPGA implementations of the SHA-3 finalists," *Proceedings - 2011 International Conference on Reconfigurable Computing and FPGAs, ReConFig 2011*, pp. 235–241, 2011.

[4] B. Jungk and M. Stottinger, "Among slow dwarfs and fast giants: A systematic design space exploration of KECCAK," *2013 8th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip, ReCoSoC 2013*, 2013.

[5] G. Bertoni, J. Daemen, M. Peeters, G. V. Assche, and R. V. Keer, "1001 Ways To Implement Keccak," *Third SHA-3 candidate conference, Washington DC*, pp. 4–8, 2012.

[6] B. Jungk, "FPGA-based Evaluation of Cryptographic Algorithms," Ph.D. dissertation, Goethe University Frankfurt, Frankfurt, 2015. [Online]. Available: http://publikationen.ub.uni-frankfurt.de/files/39388/dissertation.pdf

[7] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche, "The Keccak reference," January 2011, http://keccak.noekeon.org/.